Graphical Encoding of a Spatial Logic for the π-calculus[*]

Fabio Gadducci and Alberto Lluch Lafuente

Dipartimento di Informatica, Università di Pisa
largo Bruno Pontecorvo 3c, I-56127 Pisa, Italia
`gadducci@di.unipi.it`, `lafuente@di.unipi.it`

Abstract. This paper extends our approach to the verification of spatial properties of π-calculus specifications. The mechanism is based on a graphical encoding for mobile calculi where each process is mapped into a graph (with interfaces) such that the denotation is fully abstract with respect to the usual structural congruence, i.e., two processes are equivalent exactly when the corresponding encodings yield the same graph. Behavioral and structural properties of π-calculus processes expressed in a spatial logic are verified on the graphical encoding of a process rather than on its textual representation. For this purpose we introduce a modal logic for graphs and define a faithful translation of spatial formulae such that a process verifies a spatial formula exactly when its graphical representation verifies the translated modal graph formula.

1 Introduction

Spatial logics are formalisms for expressing behavioral and topological properties of system specifications, given as processes of a calculus. Besides the temporal modalities of the Hennessy-Milner tradition, these logics include ingredients for reasoning about the structural properties of a system. The connective $\mathbf{0}$ represents e.g. the (processes structurally congruent to the) empty system, and the formula $\phi_1|\phi_2$ is satisfied by processes that can be decomposed into two parallel components, satisfying ϕ_1 and ϕ_2, respectively. Moreover, these logics are equipped with mechanisms for reasoning about the names occurring in a system.

There are several approaches to the verification of spatial properties, on logics either for process calculi (see e.g. [4–6] and the references therein) or for other data structures such as heaps [22], trees [8] and graphs [7]. In this paper we present an approach [16] to the verification of spatial formulae [4] for π-calculus specifications, based on a graphical encoding for nominal calculi [15]. Even if a few articles have been already proposed on the verification of graphically described systems (see e.g [1, 21, 23]), to the best of our knowledge our approach is the only one that deals with specification of spatial properties for processes of nominal calculi, based on a graphical presentation. The approach was introduced in previous works, first describing the graphical encoding of processes in

[*] Research partially supported by the EU FP6-IST IP 16004 SEnSOria (*Software Engineering for Service-Oriented Overlay Computers*).

a nominal calculus [15] and then an algorithm to verify properties on such representations [16]. The present paper elaborates in the latter work, removing the restriction to finite processes and formulae and further proposing an encoding of formulae in a spatial logic for processes into formulae in a modal graph logic. Our encoding is sound and complete, i.e., a process verifies a spatial formula exactly when its graphical representation verifies the translated formula.

The main novelty of this work is likely the modal graph logic we introduce. Indeed, our first approximation to the approach was to obtain an encoding in an existing graph logic. The approaches (e.g. [2, 12, 21]) we are aware of, however, cannot properly model notions like freshness. We have thus devised a graph logic equipped with a modal operator that captures the names of those items involved in a graph transformation and that ensures that the new items to be fresh, i.e., different from any item in the formula and in the transformed graph.

Our paper provides a mechanism for specifying spatial formulae on the graphical representation of processes. We believe that our approach offers novel insights on the specification of spatial formulae, thanks especially to the link with a logics for graphs; moreover, it offers further evidence of the adequacy of graph-based formalisms for system design and specification; finally, it suggests a rich and flexible formalism for expressing properties of graph transformation.

The structure of the paper is as follows. Section 2 summarizes the π-calculus and the spatial logic for processes proposed in [4]. Sections 3 and 4 recalls the main definitions concerning graphs with interfaces [9] and their rewritings. Section 5 presents an encoding of π-calculus processes into graphs with interfaces, streamlining the proposal already discussed in [15]. Section 6 illustrates a set of graph transformation rules for simulating process reductions and assisting the encoding. Section 7 defines our modal graph logic, while Section 8 proposes the encoding of spatial formulae into graph formulae. The final section concludes the paper and outlines future research avenues.

2 The π-calculus and a Spatial Logic

This section recalls the basics of one of the foremost calculi for specifying distributed systems, namely the π-calculus [18], and of a logic [4] for expressing spatial properties of a system specified as a process of that calculus.

Definition 1 (processes). *Let $\mathcal{N}$ be a set of* names*; let $\mathcal{X}$ be a set of process variables; and let $\Delta = \{a(b), \overline{a}b \mid a, b \in \mathcal{N}\}$ be the set of* prefix operators*. A* process *P is a term generated by the syntax*

$$P ::= \ \mathtt{0} \ \mid \ (\nu a)P \ \mid \ P \mid P \ \mid \ \delta.P \ \mid \ \delta.x \ \mid \ rec_x.P$$

where $a \in \mathcal{N}$, $x \in \mathcal{X}$ and $\delta \in \Delta$. We denote by $\mathcal{P}$ the set of closed *processes, i.e., such that each process variable x occurs inside the scope of a* $\mathrm{rec}_{x}\text{\_}$ *operator.*

The standard definition for the set of free names of a process P, denoted by $\mathtt{fn}(P)$, is assumed. Similarly for α-convertibility, with respect to the *restriction* operators $(\nu a)P$ and the *input* operators $b(a).P$: In both cases, the name a is bound in P, and it can be freely α-converted.

Using the definition above, the behavior of a process P is described as a relation over *abstract processes*, i.e., a relation obtained by closing a set of basic reduction rules under structural congruence.

Definition 2 (structural congruence). *The* structural congruence *for processes is the relation* $\equiv \subseteq \mathcal{P} \times \mathcal{P}$*, closed under process construction and* α*-conversion, inductively generated by the following set of axioms*

$$P \mid Q \equiv Q \mid P \quad P \mid (Q \mid R) \equiv (P \mid Q) \mid R \quad P \mid \mathsf{0} \equiv P \quad \mathrm{rec}_x.P \equiv P\{^{rec_x.P}/_x\}$$
$$(\nu a)(\nu b)P \equiv (\nu b)(\nu a)P \qquad (\nu a)(P \mid Q) \equiv P \mid (\nu a)Q \textit{ for } a \notin \mathtt{fn}(P)$$

As usual, $P\{^Q/_x\}$ denotes process P after the substitution of each free occurrence of process variable x with process Q.

Definition 3 (reductions). *The* reduction relation *for processes is the equivalence relation* $\rightarrow \subseteq \mathcal{P} \times \mathcal{P}$*, closed under the structural congruence* $\equiv$*, inductively generated by the following set of axioms and inference rules*

$$\frac{}{a(b).P \mid \overline{a}c.Q \rightarrow P\{^c/_b\} \mid Q} \qquad \frac{P \rightarrow Q}{(\nu a)P \rightarrow (\nu a)Q} \qquad \frac{P \rightarrow Q}{P \mid R \rightarrow Q \mid R}$$

The first rule denotes the communication between two processes: Process $\overline{a}c.Q$ is ready to communicate the (possibly global) name c along channel a; it then synchronizes with process $a(b).P$, and the local name b is substituted by c on the residual process P (avoiding, as usual, the capture of name c). The latter rules state the closure of the reduction relation with respect to the operators of restriction and parallel composition.

We now recall the spatial logics for the π-calculus presented in [4].

Definition 4 (spatial logic syntax). *Let* $V_{\mathcal{N}}$ *be a set of* name variables*; and let* $V_{\mathcal{S}}$ *be a set of propositional variables. A* spatial formula *is a term generated by the syntax*

$$\phi ::= T \mid \neg\phi \mid \phi\vee\phi \mid \mathbf{0} \mid \phi|\phi \mid \eta \circledR \phi \mid \exists x.\phi \mid \text{И}x.\phi \mid \eta = \eta' \mid \Diamond\phi \mid Z \mid \mu Z.\phi$$

where $\eta, \eta' \in V_{\mathcal{N}} \uplus \mathcal{N}$*,* $x \in V_{\mathcal{N}}$ *and* $Z \in V_{\mathcal{SF}}$*. We denote by* $\mathcal{SF}$ *the set of* well-formed, name-closed *formulae, i.e., such that each propositional variable occurs inside the scope of an even number of negation operators and each name variable occurs inside the scope of a name quantifier.*

Boolean connectives and fixpoints have the usual meaning; $\mathbf{0}$ characterizes processes that are structurally congruent to the empty process; $\phi_1|\phi_2$ holds for processes that are structurally congruent to the composition of two subprocesses, satisfying ϕ_1 and ϕ_2, respectively; $\eta \circledR \phi$ is true for those processes such that ϕ holds after the revelation of name η; $\exists x.\phi$ characterizes processes such that ϕ holds for some name in $\mathcal{N}$; $\text{И}x.\phi$ holds for a process P if ϕ holds for some name of $\mathcal{N}$ that is *fresh* with respect to P and ϕ (see below); $\eta = \eta'$ requires η and η' to be equal; and $\Diamond\phi$ is satisfied by a process P if P can be reduced into Q[1] and Q satisfies ϕ.

The semantics of a (well-formed, name-closed) formula is given in terms of the domain $\mathcal{P}_{\mathcal{S}}$ of *Psets*. A Pset is a family of processes that is closed under structural congruence and name permutations, for all the names outside its *support*. Intuitively, the support for a Pset is a set of names that are relevant for the property, i.e., such that any permutation of those names outside the support does not affect the property.

Definition 5 (Pset [5]). *Let $\mathcal{Y}$ be a set of processes. Then $\mathcal{Y}$ forms a Pset if it is closed under structural congruence and there exists a finite set of names $N \subset \mathcal{N}$ such that $P\{a \leftrightarrow b\} \in \mathcal{Y}$ for all $a, b \notin N$ and $P \in \mathcal{Y}$, where $P\{a \leftrightarrow b\}$ denotes the process P after the transposition of names a and b.*

Every Pset $\mathcal{Y}$ has a least support [5, Prop. 4.13], denoted $\mathtt{supp}(\mathcal{Y})$. For instance, the set $\mathcal{P}$ of all processes is a Pset with empty support.

Formulae with open propositional variables are interpreted under an environment $\sigma : V_{\mathcal{S}} \to \mathcal{P}_{\mathcal{S}}$ which maps every open propositional variable into a Pset. The semantics of $\text{И}x.\phi$ requires x to be instantiated with a name that is fresh with respect to ϕ and to any process in the Psets to which the open propositional variables of ϕ are mapped, i.e., the name must be different from any name in ϕ or in the least support of $\sigma(Z)$ for any open propositional variable Z in ϕ. Such a set of names is defined as $\mathtt{n}_\sigma(\phi) = \mathtt{n}(\phi) \cup \bigcup_{Z \in \mathtt{fpv}(\phi)} \mathtt{supp}(\sigma(Z))$, where $\mathtt{fpv}(\phi)$, $\mathtt{n}(\phi)$ and $\mathtt{supp}(\mathcal{Y})$ respectively denote the set of the free propositional variables of ϕ, the set of names of ϕ, and the least support of $\mathcal{Y}$.

Definition 6 (spatial logic semantics). *Let ϕ be a (well-formed, name-closed) spatial formula and let σ be a mapping for the free propositional variables of ϕ into Psets. The denotation $[\![\phi]\!]_\sigma$, mapping a formula ϕ into a Pset, is defined by structural induction according to the following rules*

$$\begin{array}{rlrl}
[\![T]\!]_\sigma &= \mathcal{P} & [\![a \circledR \phi]\!]_\sigma &= \{P \mid \exists P'.P \equiv (\nu a)P' \textit{ and } P' \in [\![\phi]\!]_\sigma\} \\
[\![\neg\phi]\!]_\sigma &= \mathcal{P} \setminus [\![\phi]\!]_\sigma & [\![\exists x.\phi]\!]_\sigma &= \bigcup_{a \in \mathcal{N}} [\![\phi\{^a/_x\}]\!]_\sigma \\
[\![\phi_1 \vee \phi_2]\!]_\sigma &= [\![\phi_1]\!]_\sigma \cup [\![\phi_2]\!]_\sigma & [\![\text{И}x.\phi]\!]_\sigma &= \bigcup_{a \notin \mathtt{n}_\sigma(\phi)} ([\![\phi\{^a/_x\}]\!]_\sigma \setminus \{P \mid a \in \mathtt{fn}(P)\}) \\
[\![\mathbf{0}]\!]_\sigma &= \{P \mid P \equiv 0\} & [\![a = b]\!]_\sigma &= \mathcal{P} \textit{ if } a = b \textit{ and } \emptyset \textit{ otherwise} \\
[\![Z]\!]_\sigma &= \sigma(Z) & [\![\mu Z.\phi]\!]_\sigma &= \mathtt{lpf}(\lambda Y.[\![\phi]\!]_{\sigma[Y/Z]}) \\
[\![\phi_1 | \phi_2]\!]_\sigma &= \multicolumn{3}{l}{\{P \mid \exists P_1, P_2.P \equiv P_1 | P_2 \textit{ and } P_1 \in [\![\phi_1]\!]_\sigma \textit{ and } P_2 \in [\![\phi_2]\!]_\sigma\}} \\
[\![\Diamond\phi]\!]_\sigma &= \multicolumn{3}{l}{\{P \mid \exists Q.P \to Q \textit{ and } Q \in [\![\phi]\!]_\sigma\}}
\end{array}$$

where $\mathtt{lpf}(f)$ denotes the least fixed-point of the function f.

The restriction on the use of negation guarantees each possible function $\mathtt{lpf}(\lambda Y.[\![\phi]\!]_{\sigma[Y/Z]})$ to be monotonic, so that fixed points are well defined. Indeed, its semantics coincides with $\bigcap_{\mathcal{Y} \in \mathcal{P}_s | \mathcal{Y} \subseteq [\![\phi]\!]_{\sigma[\mathcal{Y}/Z]}}$ [4].

[1] For the sake of brevity, the unique action modality is synchronization: Our approach can be easily extended to handle commitments as defined in [4].

3 Graphs and their extension with interfaces

We recall a few definitions concerning (typed hyper-)graphs, and their extension with *interfaces*, referring to [3, 9] for a more detailed introduction.

Definition 7 (graphs). *A(n hyper-)*graph *is a four-tuple $\langle V, E, s, t\rangle$ where V is the set of nodes, E is the set of edges and $s, t : E \to V^*$ are the source and target functions. A(n hyper-)graph morphism is a pair of functions $\langle f_V, f_E\rangle$ preserving the source and target functions, i.e., $f_V \circ s = s \circ f_E$ and $f_V \circ t = t \circ f_E$.*

However, we shall consider *typed graphs* [10], i.e., graphs labeled over a structure that is itself a graph.

Definition 8 (typed graphs). *Let T be a graph. A* typed graph *G over T is a graph $|G|$, together with a graph morphism $\tau_G : |G| \to T$. A* morphism *between T-typed graphs $f : G_1 \to G_2$ is a graph morphism $f : |G_1| \to |G_2|$ consistent with the typing, i.e., such that $\tau_{G_1} = \tau_{G_2} \circ f$.*

In the following, a chosen type graph T is assumed.

In order to inductively define the encoding for processes, we need to provide operations over typed graphs. The first step is to equip them with suitable "handles" for interacting with an environment.

Definition 9 (graphs with interfaces). *A (T-typed)* graph with interfaces *(shortly,* GWI*) is a triple $\mathbb{G} = \langle i_G, G, o_G\rangle$, for G a T-typed graph and $i_G : I_G \to G$, $o_G : O_G \to G$ the* input *and* output *graph morphisms.*

An interface graph morphism *$f : \mathbb{G} \Rightarrow \mathbb{G}_2$ is a triple of graph morphisms $\langle f_I, f, f_O\rangle$, with f_I, f_O injective, preserving the input and output morphisms.*

The category of T-typed graphs with interfaces is denoted by I-T-**Graph**. We let $I \xrightarrow{i} G \xleftarrow{o} O$ denote a graph (body) with *input* interface I and *output* interface O. With an abuse of notation, we sometimes refer to the image of the input and output morphisms as inputs and outputs, respectively. More importantly, in the following we often refer implicitly to a GWI as the representative of its isomorphism class, still using the same symbols to denote it and its components.

In order to define our process encoding, we introduce two operators on graphs with *discrete* interfaces (GWDIs), i.e., such that their set of edges is empty.

Definition 10 (two operators). *Let $\mathbb{G} = I \xrightarrow{i} G \xleftarrow{j} J$ and $\mathbb{G}' = J \xrightarrow{j'} G' \xleftarrow{o} O$ be* GWDI*s. Then, their* sequential composition *is the* GWDI *$\mathbb{G} \circ \mathbb{G}' = I \xrightarrow{i'} G'' \xleftarrow{o'} O$, for G'' the disjoint union $G \uplus G'$, modulo the equivalence on nodes induced by $j(x) = j'(x)$ for all $x \in N_J$, and i', o' the uniquely induced arrows.*

Let $\mathbb{G} = I \xrightarrow{i} G \xleftarrow{o} O$ and $\mathbb{H} = I' \xrightarrow{i'} H \xleftarrow{o'} O'$ be GWDI*s with compatible interfaces.*[2] *Then, their* parallel composition *is the* GWDI *$\mathbb{G} \otimes \mathbb{H} = (I \cup I') \xrightarrow{i''}$*

[2] That is, any node in $N_I \cap N_{I'}$ has the same type in I and I' (similarly for $N_O \cap N_{O'}$).

$G' \xleftarrow{o''} (O \cup O')$, for G' the disjoint union $G \uplus H$, modulo the equivalence on nodes induced by $o(y) = o'(y)$ for all $y \in N_O \cap N_{O'}$ and $i(y) = i'(y)$ for all $y \in N_I \cap N_{I'}$, and i'', o'' the uniquely induced arrows.

With an abuse of notation, the set-theoretic operators on graphs are defined component-wise, and the typing morphism is extended accordingly. Intuitively, the sequential composition $\mathbb{G} \circ \mathbb{G}'$ is obtained by taking the disjoint union of the bodies of $\mathbb{G}$ and $\mathbb{G}'$, and gluing the outputs of $\mathbb{G}$ with the corresponding inputs of $\mathbb{G}'$. Similarly, the parallel composition $\mathbb{G} \otimes \mathbb{H}$ is obtained by taking the disjoint union of the bodies of $\mathbb{G}$ and $\mathbb{H}$, additionally gluing the inputs (outputs) of $\mathbb{G}$ with the corresponding inputs (outputs) of $\mathbb{H}$. The two operations are defined on "concrete" graphs, even if the result is independent of the choice of the representatives, up-to isomorphism[3].

A *graph expression* is a term over the syntax containing all graphs with discrete interfaces as constants, and parallel and sequential composition as binary operators. An expression is *well-formed* if all occurrences of those operators are defined for the interfaces of their arguments, according to Definition 10; its interfaces are computed inductively from the interfaces of the graphs occurring in it, and its *value* is the graph obtained by evaluating all operators in it.

4 Rewriting graphs with interfaces

This section recalls the basic tools of the double-pushout (DPO) approach to (typed hyper-)graph transformation, as presented in [11, 13]. More precisely, it directly introduces the extension of the approach to GWIs, which is needed later on for our modeling purposes.

Definition 11 (graph production). *A* graph production *is a pair of arrows $\langle l : \mathbb{K} \to \mathbb{L}, r : \mathbb{K} \to \mathbb{R}\rangle$ in I-T-**Graph** such that the three components of l are injective. A T-typed* graph transformation system *(GTS) $\mathcal{G}$ is a tuple $\langle T, P, \pi\rangle$ where T is the type graph, P is a set of production names and π is a function mapping each name to a T-typed production.*

A production $\pi(p)$ is usually denoted by a *span* $\mathbb{L} \xleftarrow{l} \mathbb{K} \xrightarrow{r} \mathbb{R}$, and it is often indicated just by the name p. Usually, l is denoted as being a monomorphism.

Definition 12 (derivation). *Let $p : \mathbb{L} \xleftarrow{l} \mathbb{K} \xrightarrow{r} \mathbb{R}$ be a T-typed production and $\mathbb{G}$ a T-typed GWI. A* match *of p in $\mathbb{G}$ is a morphism $m_{\mathbb{L}} : \mathbb{L} \to \mathbb{G}$. A* direct

[3] While the sequential operator corresponds to categorical composition, the parallel operator only recalls the tensor product of monoidal categories. A more standard definition for the latter operator can be found in [9]. Our choice, though, allows for a compact presentation of the graphical encoding in the following sections.

derivation *from* $\mathbb{G}$ *to* $\mathbb{H}$ *via production* p *at a match* $m_{\mathbb{L}}$ *is a diagram*

$$\begin{tikzcd}
p: & \mathbb{L} \arrow[d, "m_L"'] \arrow[r, phantom, "(1)", shift right=4ex] & \mathbb{K} \arrow[l, "l"'] \arrow[r, "r"] \arrow[d, "m_K"] \arrow[r, phantom, "(2)", shift right=4ex] & \mathbb{R} \arrow[d, "m_R"] \\
& \mathbb{G} & \mathbb{D} \arrow[l, "l^*"] \arrow[r, "r^*"'] & \mathbb{H}
\end{tikzcd}$$

where (1) and (2) are actually pushout squares in I*-*T*-***Graph***. We thus write* $p/m : \mathbb{G} \Longrightarrow \mathbb{H}$*, for* m *the morphism* $\langle m_L, m_K, m_R \rangle$*, or simply* $\mathbb{G} \Longrightarrow \mathbb{H}$*.*

Operationally, the application of a production p to a GWI $\mathbb{G}$ consists of three steps, performed component-wise. Consider, e.g., the bodies of the GWIs. First, the match $m_L : L \to G$ is chosen, providing an occurrence of L in G. Then, all the items of G matched by $L - l(K)$ are removed, leading to the context graph D. If D is well-defined, and the resulting square is indeed a pushout, the items of $R - r(K)$ are added to D, further coalescing those nodes and edges identified by r, obtaining the derived graph H.

Let p be a production, let $p/m : \mathbb{G} \Longrightarrow \mathbb{H}$ be a direct derivation and let $tr(p/m)$ be the partial function $r^* \circ (l^*)^{-1} : \mathbb{G} \to \mathbb{H}$. By construction, $tr(p/m)$ is injective on interfaces. The derivation is *interface preserving* if $tr(p/m)$ actually preserves node identity on interfaces. From now on, we will restrict our attention to derivations that are interface preserving.

5 From Processes to Graphs

We now present an encoding of π-calculus processes into graphs with interfaces, based on the encoding introduced in [15].

The type graph is defined in Fig. 1. Note that all edges have at most one node in the source, connected by an incoming tentacle; the nodes in the target list are instead always enumerated clock-wise, starting from the only incoming tentacle, unless otherwise specified by an enumerating label. For example, the edge ν has the node $\bullet$ as source, and the node $\circ$ as target. The edge *op* actually stands as a concise representation for two edges, namely *in* and *out*, with the same source and target: they have the node $\bullet$ as source and the node list $\langle \bullet, \circ, \circ \rangle$ as target, further specified by the enumerating labels 0, 1, and 2.

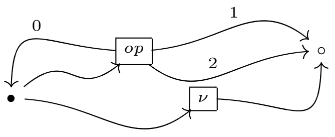

Fig. 1. The type graph (for $op \in \{in, out\}$).

The type graph is used to model processes syntactically, and our encoding corresponds to the usual construction of the tree associated to a term of an algebra: Names are interpreted as variables, so that they are mapped to leaves of the tree and can be safely shared. Intuitively, a tree with a node of type $\bullet$ as root corresponds to a process, whilst each node of type $\circ$ basically represents a name. Clearly, the operators *in* and *out* simulate the input and output prefixes, respectively; and operator ν stands for restriction. Furthermore, note that there is instead no explicit operator accounting for parallel composition.

The second step is the characterization of a class of graphs, such that all processes can be encoded into a graph expression. Let $p \notin \mathcal{N}$: Our choice is depicted in Fig. 2, for all $a, b \in \mathcal{N}$.

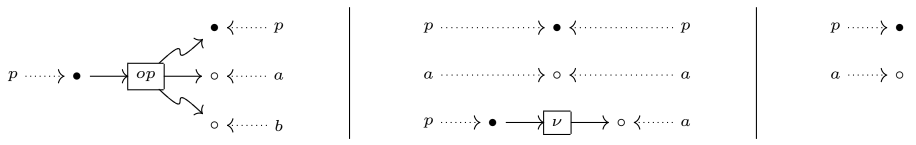

Fig. 2. Graphs $op_{a,b}$ (for $op \in \{in, out\}$); id_p, id_a, and ν_a; $\mathtt{0}_p$ and $\mathtt{0}_a$.

Finally, let us denote id_Γ and $\mathtt{0}_\Gamma$ as a shorthand for $\bigotimes_{a\in\Gamma} id_a$ and $\bigotimes_{a\in\Gamma} \mathtt{0}_a$, respectively, for a finite set of names $\Gamma \subset \mathcal{N}$ (since the ordering is immaterial). The encoding of finite processes into GWDIs, mapping each finite process into a graph expression, is presented below.

Definition 13 (encoding for finite processes). *Let P be a finite process, and let Γ be a set of names, such that $\mathtt{fn}(P) \subseteq \Gamma$. The* process encoding $\llbracket P \rrbracket_\Gamma$, *mapping a process P into a* GWDI, *is defined by structural induction according to the following rules (where $\{c\} \uplus \Gamma$ implies that $c \notin \Gamma$)*

$$\llbracket (\nu a)P \rrbracket_\Gamma = \begin{cases} \llbracket P \rrbracket_\Gamma & \text{if } a \notin \mathtt{fn}(P) \\ (id_p \otimes \nu_c \otimes id_\Gamma) \circ \llbracket P\{^c/_a\} \rrbracket_{\{c\}\uplus\Gamma} & \text{otherwise} \end{cases}$$

$$\llbracket P \mid Q \rrbracket_\Gamma = \llbracket P \rrbracket_\Gamma \otimes \llbracket Q \rrbracket_\Gamma \qquad \llbracket a(b).P \rrbracket_\Gamma = (in_{a,c} \otimes id_\Gamma) \circ \llbracket P\{^c/_b\} \rrbracket_{\{c\}\uplus\Gamma}$$

$$\llbracket \mathtt{0} \rrbracket_\Gamma \quad = \mathtt{0}_p \otimes \mathtt{0}_\Gamma \qquad \llbracket \overline{a}b.P \rrbracket_\Gamma \quad = (out_{a,b} \otimes id_\Gamma) \circ \llbracket P \rrbracket_\Gamma$$

Note the conditional rule for $(\nu a).P$: It is required for removing the occurrence of useless restriction operators, i.e., those binding a name not occurring in the process. The mapping is well-defined, since the resulting graph expression is well-formed, and the encoding $\llbracket P \rrbracket_\Gamma$ is a graph with interfaces $(\{p\} \cup \Gamma, \emptyset)$.

The mapping $\llbracket \cdot \rrbracket$ is not surjective, since there are graphs of rank $(\{p\} \cup \Gamma, \emptyset)$ that are not (isomorphic to) the image of any process. Nevertheless, our encoding is sound and complete, as stated by the proposition below (adapted from [15]).

Proposition 1 (correct process encoding). *Let P, Q be finite processes and let Γ be a set of names such that $\mathtt{fn}(P) \cup \mathtt{fn}(Q) \subseteq \Gamma$. Then, $P \equiv Q$ if and only if $\llbracket P \rrbracket_\Gamma = \llbracket Q \rrbracket_\Gamma$.*

In order to show how recursive processes can be encoded as suitable infinite graphs, the first step is to consider a complete partial order on graphs.

Definition 14 (graph order). *Let $\mathbb{G}$, $\mathbb{H}$ be* GWDI*s with interfaces (I, O). Then, $\mathbb{G} \sqsubseteq_{I,O} \mathbb{H}$ if there exists an injective graph morphim $f : \mathbb{G} \to \mathbb{H}$.*

Thus, we consider the standard subgraph relationship, partitioned over interfaces: often, we skip subscripts whenever clear from the context. These partial orders are complete with respect to ω-chains, and it is noteworthy that the encoding $\llbracket \mathtt{0} \rrbracket_{\Gamma}$ is the bottom of the order for those GWDIs with interfaces $(\{p\} \cup \Gamma, \emptyset)$.

Definition 15. *Let $rec_x.P$ be a process, and let Γ be a set of names, such that $\mathtt{fn}(P) \subseteq \Gamma$. Then, the encoding $\llbracket rec_x.P \rrbracket_{\Gamma}$ is defined as $\mathtt{lpf}(\lambda X. \llbracket P\{^X/_x\} \rrbracket_{\Gamma})$.*

Two recursive processes may be mapped to isomorphic GWDIs, even if they are not structurally congruent. Nevertheless, the extended encoding is still sound.

6 Process reductions vs graph rewrites

This section introduces a rule for simulating the reduction relation as well as a few rules that are useful for the encoding of the logic. In the following, even if not explicitly stated, all the spans involve only graphs with empty output interfaces.

So, let us start with rule p_π (depicted in Fig. 3) for simulating the reduction relation over processes given in Definition 3.

Let us explain our notation. The nodes may be labeled. If the label is an element in $\{p\} \cup \mathcal{N}$, that means that the node is actually in the image of the input interface. Otherwise, the label is a natural number, and it is used just for describing the actions performed by the rule, so that e.g. the $\circ$ nodes identified by 2 and 3 are coalesced by the rule. These identifiers are of course arbitrary: They correspond to the actual elements of the set of nodes/interfaces, and they unambiguously characterise the (interface preserving) span of functions.

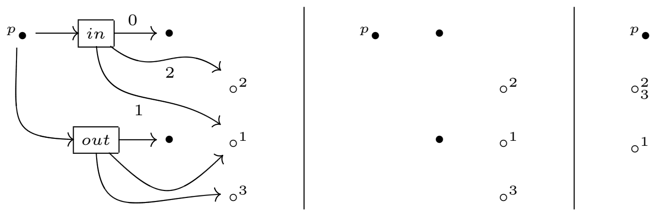

Fig. 3. The rule p_π for synchronization.

It is noteworthy that just one rule is needed to recast the reduction semantics for the π-calculus. The structural rules are taken care of by the fact that graph morphisms allow for embedding a graph into a larger one, thus simulating the closure of reduction with respect to contexts. Similarly, no distinct instance of the rule is needed, since graph isomorphism takes care of the closure with respect to structural congruence, and of the renaming of free names.

We now introduce a set of "house-keeping" rules for performing specific tasks requested by our encoding of the spatial logic. The rule p_n for adding nodes to the interface is depicted on the left of Fig. 4. Since the left-most and middle graphs are empty, the rule can be applied to any graph resulting in the addition of a node to the right-most graph. This rule is going to be used in conjunction with rule p_r: It reveals a restricted name (see Fig. 5), consuming a restriction-edge and coalescing the attached node with the image of an interface node.

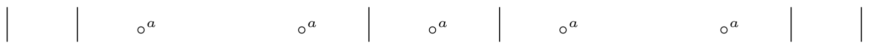

Fig. 4. Rules for introducing (left), checking for (center) and removing a name (right).

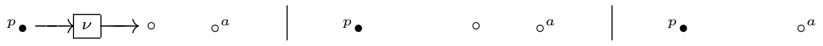

Fig. 5. The rule for revealing a restricted name.

The identity rule $p_{\exists_i}$, represented on the right of Fig. 4, is used to test the presence of a node among the inputs. We do not depict the similar rule $p_{\exists_b}$, without the node a in the interface, testing for the presence of any node.

Finally, the garbage collection rule p_g depicted on the center of Fig. 4 is used to remove a name from the interface: Note that the DPO formalism ensures that the rule is applied properly, i.e., to an isolated node only.

7 Modal Graph Logic

This section introduces our flavor of graph logic, inspired by [2, 12, 21] and resulting in a monadic second order μ-calculus with a first-order action modality. In particular, our logic is closely related to [2], where a fragment of Courcelle's monadic second order logic [12], combined with the propositional μ-calculus, is considered. The main novelty of our proposal is a first-order action modal operator that requires a rewriting via a given rule to exist and binds a set of variables with the identities of the nodes involved in the corresponding derivation.

Definition 16 (graph logic syntax). *Let V_Z be a set of propositional variables, V_n a set of node variables, V_e, V_E set of first and second order edge variables, respectively, and finally $\langle T, P, \pi\rangle$ a* GTS *The set $\mathcal{GF}$ of all* graph formulae *over the* GTS $\langle T, P, \pi \rangle$ *is the set of terms generated by*

$$\psi ::= T \mid \theta \mid \neg\psi \mid \psi \vee \psi \mid \langle p(\boldsymbol{x}, \boldsymbol{x'})\rangle\psi \mid \exists_s x.\psi \mid \exists y.\psi \mid \exists Y.\psi \mid Z \mid \mu Z.\psi$$
$$\theta ::= \epsilon = \epsilon \mid y = y \mid \tau(y) = t_e \mid y \in Y \qquad \epsilon ::= \eta \mid i(\eta) \mid s(y) \mid t[k](y)$$

where $k \in \mathbb{N}$, $x \in V_n$, $\eta \in V_n \cup \mathcal{N}$, $y \in V_e$, $\boldsymbol{x}, \boldsymbol{x'} \in V_n^*$, $Y \in V_E$, $Z \in V_Z$, $t_e \in E_T$, *and* $p \in P$.

For readability sake, the rule p in the above definition is interface preserving, and $\boldsymbol{x}$, $\boldsymbol{x'}$ are vectors of node variables indexed over the nodes of the left-hand side $\mathbb{L}$ and of the right-hand side $\mathbb{R}$, respectively.

As we shall see, the modal operator is used to bind variables with the identities of items matched in both the left- and the right-hand side of a rule. Thus, the modal operator $p(\langle x_1, \ldots, x_n\rangle, \langle x'_1, \ldots, x'_m\rangle).\phi$ bounds the $n + m$ variables in ϕ. In the following we consider closed formulae only, i.e., formulae where each occurrence of a node, edge, edge set or propositional variable is bound.

The logic includes booleans, a first-order node quantifier, first and second-order edge quantifiers, a modal operator, fixpoints, and equalities of edge identities or nodes (possibly referred to by node variables), the source or i-th target of an edge, or the images of an input, denoted by η, $s(y)$, $t[k](y)$, and $i(\eta)$ respectively. Note the lack of constraints on the number of tentacles departing from an edge variable, so that a formula as $\exists n.t[i](e) = n$ might turn out to be false.

We introduce now the concept of Gsets, sets of GWDIs closed under graph isomorphism and permutations of interface nodes outside its *support.*

Definition 17 (Gset). *Let $\mathcal{Y}$ be a set of* GWDI*s. Then $\mathcal{Y}$ forms a Gset if there exists a finite set of interface nodes N such that $f(\mathbb{G}) \in \mathcal{Y}$ for all* GWI*s $\mathbb{G} \in \mathcal{Y}$ and isomorphisms f being identities on the nodes in N.*

Each Gset $\mathcal{Y}$ can be proved to have a finite support, denoted by $\mathtt{supp}(\mathcal{Y})$; and clearly (the union of) the encoding of (the members of) a Pset turns out to be a Gset. We let $\mathtt{n}_\rho(\psi)$ denote the set of interface names of a formula ψ under a valuation ρ defined as $\mathtt{n}(\psi) \cup \bigcup_{Z \in \mathtt{fpv}(\psi)} \mathtt{supp}(\rho(Z))$, where $\mathtt{fpv}(\psi)$ and $\mathtt{n}(\psi)$ respectively denote the set of the free propositional variables and the names of a formula ψ (constants and free name variables).

The formulae of the logic are intended to be interpreted over Gsets.

Definition 18 (graph logic semantics). *Let ψ be a graph formula and let ρ be a 4-tuple $\langle \rho_x, \rho_y, \rho_Y, \rho_Z\rangle$ of mappings from node, edge, edge set and propositional variables into nodes, edges, edge sets, and Gsets, respectively. The denotation $[\![\psi]\!]_\rho$, mapping a formula ψ into a Gset, is defined by structural induction according to the following rules*

$$\begin{array}{rlrl}
[\![T]\!]_\rho &= S & [\![\exists y.\psi]\!]_\rho &= \{\mathbb{G} \in S \mid \exists e \in E_\mathbb{G}.\mathbb{G} \in [\![\psi]\!]_{\rho[e/y]}\} \\
[\![\neg\psi]\!]_\rho &= S \setminus [\![\psi]\!]_\rho & [\![\exists Y.\psi]\!]_\rho &= \{\mathbb{G} \in S \mid \exists E \subseteq E_\mathbb{G}.\mathbb{G} \in [\![\psi]\!]_{\rho[E/Y]}\} \\
[\![Z]\!]_\rho &= \rho_Z(Z) & [\![\mu Z.\psi]\!]_\rho &= \mathtt{lpf}(\lambda v.[\![\psi]\!]_{\rho[v/Z]}) \\
[\![\theta]\!]_\rho &= \|\rho(\theta)\| & [\![p(\boldsymbol{x}, \boldsymbol{x'})\psi]\!]_\rho &= \{\mathbb{G} \mid p/m : \mathbb{G} \to \mathbb{H} \textit{ and } \mathbb{H} \in [\![\psi]\!]_{\rho'}\} \\
[\![\psi_1 \vee \psi_2]\!]_\rho &= [\![\psi_1]\!]_\rho \cup [\![\psi_2]\!]_\rho & [\![\exists_s x.\psi]\!]_\rho &= \bigcup_{a \in n_\rho(\psi)} [\![\psi]\!]_{\rho[^a/_x]}
\end{array}$$

where $\mathtt{lpf}(f)$ *denotes the least fixed-point of the function* f*,* $\|\theta\|$ *maps true and false to* S_M *and* $\emptyset$*, respectively, and* $\rho' = tr^{\dagger}(p/m) \circ (\rho \cup \{\boldsymbol{x} \mapsto m(N_{\mathbb{L}})\}) \cup \{\boldsymbol{x'} \mapsto m(N_{\mathbb{R}})\}$*, for* $N_{\mathbb{L}}$ *and* $N_{\mathbb{R}}$ *the nodes of the left-hand side* $\mathbb{L}$ *and the righ-hand side* $\mathbb{R}$ *of the rule* p*, and* $tr^{\dagger}(p/m)$ *the total extension of* $tr(p/m)$*.*

Intuitively, the variables in $\boldsymbol{x}$ are assigned to the matched items of the left-hand side of the derivation, and the resulting mapping is composed with the trace of the derivation to get rid of item renaming (no renaming is needed instead for $\boldsymbol{x'}$). In addition, we require new interface items in $\mathbb{H}$ to be different from the interface names of ψ, i.e., not to occur in $\mathtt{n}_{\rho}(\psi)$: This ensures the new items to be fresh with respect to the formula and its environment.

Boolean connectives and item comparisons have the expected meaning, and, since the denotation is for closed formulae, the interpretation of the terms generated by θ is obvious. Note however that as in [2] we consider environments ρ that might map a variables into items that are not part of some graphs. Thus, a formula like $x = y$ is satisfied by a graph in environment ρ if $\rho(x) = \rho(y)$, independently, hence, on whether or not $\rho(x)$ or $\rho(y)$ are nodes of the graph.

Indeed, the main difference with the approach of [2] is the semantics of the modal operator. In order for $p(\boldsymbol{x}, \boldsymbol{x'})\psi$ to hold in an environment ρ we require a direct derivation from $\mathbb{G}$ into a graph $\mathbb{H}$ via rule p and match m to exist such that $\mathbb{H}$ fulfills ψ in an environment ρ' that is like ρ after applying the trace of the derivation (to get rid of item renaming) and the addition of the mapping of variables in vectors $\boldsymbol{x}$, $\boldsymbol{x'}$ with the items of the left- and right-hand side of the match of rule p. In that way, one can express not only the possibility of applying a graph transformation rule, but we can bind variables with the items involved in the transformation which we can use in the residual formula.

8 From Spatial to Graph Logic

We can now finally turn our attention to the encoding $[\cdot] : \mathcal{SF} \to \mathcal{GF}$, mapping spatial formulae into graph formulae. Our goal is to define a complete and sound encoding such that for any process P we have that $P \in [\![\phi]\!]$ iff $\llbracket P \rrbracket \in [\![[\phi]]\!]$.

For the sake of readability, for each modal operator we consider only those arguments that are relevant for the encoding. So, $p_{\pi}(x_1, x_2)$ bounds x_1 and x_2 with the items 1 and 2 of the left-hand side of the rule p_{π} (see Fig. 3), i.e., the channel on which synchronization occurs and the sent name. These nodes are relevant for the encoding since they might become isolated and thus need to be garbage collected. Similarly, $p_n(x_a)$ bounds x_a with the item a of the right-hand side of the rule p_n (see Fig. 4, right), i.e., the new interface node; $p_{\exists_i}(x_a)$ bounds x_a with the item a of the rule $p_{\exists_i}$ (see Fig. 4, center), i.e., the checked for interface node (while $p_{\exists_b}(x)$ bounds x with the only node of the rule); $p_g(x_a)$ bounds x_a with the item a of the right-hand side of rule p_g (see Fig. 4, right), i.e., of the deleted interface node; and $p_r(x_a)$ bounds x_a with the item a of the right-hand side of rule p_g (see Fig. 5), i.e., the revealed interface node.[4]

[4] Note that, except for $p_{\pi}(x_1, x_2)$ and $p_{\exists_b}(x)$, all the operators bounds interface nodes, since they are used for checking name properties.

$$\begin{aligned}
\exists y \in Y.\psi &\equiv \exists y.(y \in Y \wedge \psi)\\
in(x,Y) &\equiv \exists y \in Y.(s(y) = x \vee t[0](y) = x \vee t[1](y) = x \vee t[2](y) = x)\\
x \in Y &\equiv \langle\langle p_{\exists_b}(x)\rangle\rangle in(x,Y) \;\vee\; \langle\langle p_{\exists_i}(x)\rangle\rangle in(i(x),Y)\\
\{\exists y.\psi\}^Y &\equiv \exists y \in Y.\{\psi\}^Y\\
\{\exists Y.\psi\}^Y &\equiv \exists Y'.(\forall y. y \in Y' \rightarrow y \in Y) \wedge \{\psi\}^Y\\
\{\exists_s x.\psi\}^Y &\equiv \exists_s x.i(x) \in Y \wedge \{\psi\}^Y\\
\{\langle p(\boldsymbol{x},\boldsymbol{x'})\rangle.\psi\}^Y &\equiv \langle p(\boldsymbol{x},\boldsymbol{x'})\rangle(\textstyle\bigwedge_{x\in\boldsymbol{x}\cup\boldsymbol{x'}} x \in Y \wedge \{\psi\}^Y)\\
I(x) &\equiv \neg\exists Y.x \in Y\\
S(y,y') &\equiv s(y') = t[0](y)\\
C(Y) &\equiv \langle p_{\exists_b}(x)\rangle(x = i(p) \wedge \forall y.(y \in Y \rightarrow x = s(y) \vee \exists y' \in Y.S(y',y)))\\
R(Y,Y') &\equiv \forall y.(\ \tau(y) = \nu \rightarrow (t[0](y) \in Y \leftrightarrow t[0](y) \notin Y')\ )\\
P(Y,Y') &\equiv \forall y.(y \in Y \leftrightarrow y \notin Y')\\
\langle\langle p(\boldsymbol{x_1},\boldsymbol{x'_1})\rangle\rangle\psi &\equiv \langle p(\boldsymbol{x},\boldsymbol{x'})\rangle(\boldsymbol{x} = \boldsymbol{x_1} \wedge \boldsymbol{x'} = \boldsymbol{x'_1} \wedge \psi\{{}^{\boldsymbol{x}}/_{\boldsymbol{x_1}}, {}^{\boldsymbol{x'}}/_{\boldsymbol{x'_1}}\})
\end{aligned}$$

Fig. 6. Auxiliary graph formulae.

Fig. 8 summarizes some additional abbreviations that provide a more readable and concise presentation of the encoding. First, as a shorthand, $\exists y \in Y.\psi$ quantifies over the edges of an edge set, while $in(x,Y)$ is a shorthand for the formula expressing the occurrence of the node x in either the source or the target of an edge in Y. Since the type graph considers at most three targets, the formula considers only up to the third target. Similarly, $x \in Y$ states that x is in either the source or the target of an edge in Y. Furthermore, $\{\psi\}^Y$ denotes the formula ψ relativized to the set of edges Y. Fig 8 defines the most significant cases for ψ, the others are recursively defined in a straightforward way. Formula $I(x)$ states that x is not the source or target of any edge, thus characterizing isolated nodes. Formula $S(y,y')$ states that edges y,y' occur consecutively. This can happen in encoded processes only if the source of one of the edges to be equal to the first target of the other edge. Another property is that a set of edges (in an acyclic graph, as those representing processes) is connected: In words, $C(Y)$ requires each edge of set Y to occur consecutively to another edge of Y unless it has the root of the graph (the image of p) as source. Then $R(Y,Y')$ states the confinement of the target of a restriction operator, i.e., the target of each ν edge must to be either in Y or in Y'. We also use a formula $P(Y,Y')$ to express that two sets of edges Y,Y' are disjoint and complementary, i.e., they partition the set of edges. Another abbreviation is that we sometimes want to express the fact that a rule p can be applied for a certain match denoted by $\langle\langle p(\boldsymbol{x_1},\boldsymbol{x'_1})\rangle\rangle\phi$.

We finally present our encoding of spatial formulae into graph formulae

Definition 19 (Logics encoding). *Let ϕ be a spatial formula. The* logics encoding *$[\phi]$, mapping a spatial formula ϕ into a graph formula, is defined by structural induction according to the rules in Fig. 7.*

The encoding of boolean connectives (b1) and fixpoints (μ1) is trivial.

$$\begin{array}{rll}
[\mathbf{0}] = & \forall y.F & (v1)\\
[T] = & T \qquad [\neg\phi] = \neg[\phi] \qquad [\phi_1 \vee \phi_2] = [\phi_1] \vee [\phi_2] & (b1)\\
[\eta_1 = \eta_2] = & (\eta_1 = \eta_2) & (n1)\\
[Z] = & Z \qquad [\mu Z.\phi] = \mu Z.[\phi] & (\mu 1)\\
[\text{И} x.\phi] = & \langle p_n(x)\rangle[\phi] & (f1)\\
[\exists x.\phi] = & \quad [\text{И} x.\phi] & (e1)\\
 & \vee\ \langle\langle p_{\exists_i}(x)\rangle\rangle[\phi] & (e2)\\
 & \vee\ \exists_s x.[\phi] & (e3)\\
[\eta Ⓡ \phi] = & \quad (\langle\langle p_{\exists_i}(\eta)\rangle\rangle I(i(\eta)) \wedge \langle\langle p_r(\eta)\rangle\rangle[\phi]) & (r1)\\
 & \vee\ (\neg\langle\langle p_{\exists_i}(\eta)\rangle\rangle T \wedge \langle p_n(x')\rangle\langle\langle p_r(x')\rangle\rangle[\phi\{^{x'}/_{\eta}\}]) & (r2)\\
[\Diamond\phi] = & \langle p_\pi(x,x')\rangle(I(x) \wedge I(x') \wedge \langle\langle p_g(x)\rangle\rangle\langle\langle p_g(x')\rangle\rangle[\phi]) & (a1)\\
 & \qquad\qquad \vee\ (I(x) \wedge \neg I(x') \wedge \langle\langle p_g(x)\rangle\rangle[\phi]) & (a2)\\
 & \qquad\qquad \vee\ (\neg I(x) \wedge I(x') \wedge \langle\langle p_g(x'\rangle\rangle[\phi]) & (a3)\\
 & \qquad\qquad \vee\ (\neg I(x) \wedge \neg I(x') \wedge [\phi])) & (a4)\\
[\phi_1|\phi_2] = & \exists Y.\exists Y'.P(Y,Y') & (c1)\\
 & \qquad\quad \wedge\ C(Y) \wedge C(Y') & (c2)\\
 & \qquad\quad \wedge\ R(Y,Y') & (c3)\\
 & \qquad\quad \wedge\ \{[\phi_1]\}^{Y} \wedge \{[\phi_2]\}^{Y'} & (c4)
\end{array}$$

Fig. 7. The encoding of spatial formulae into graph formulae.

Regarding the encoding of name equalities, it is worth noticing that the encoding works with interface nodes rather than with their images. Because the input morphism is injective in any process encoding, we can safely encode name comparison as comparison of the corresponding interface nodes (n1).

The encoding of an empty process is the graph 0_p depicted in Fig. 2, i.e., it is a graph with just one node and no edges. Moreover, no other GWI modeling a process has an empty set of edges. Thus, the encoding of $\mathbf{0}$ is a graph formula that characterizes graphs without edges (v1).

The encoding of the freshness quantifier exploits Gabbay-Pitts property [14]: It suffices to consider just one fresh name, neither occurring in ϕ nor previously in the process. We obtain such a name via the freshness rule p_n, which bounds the variable x as the fresh name it is introduced. The rule ensures that x will be effectively fresh for the process and the formula (f1).

Also the encoding of $\exists x.\phi$ relies on Gabbay-Pitts property. Indeed, to check if ϕ holds for some name x it suffices to consider (e1) a fresh name (thus relying on the encoding of freshness quantification), (e2) all the free names of the process (the nodes of its interface) and (e3) the nodes in $n_\sigma([\phi])$ (i.e., all the names of ϕ plus the least support of $\sigma(Z)$ for any open propositional variable Z in ϕ).

The encoding of $\eta Ⓡ \phi$ distinguishes two cases: Either (r1) the node η occurs in the interface and its image is isolated or (r2) it is not in the interface. The first turns out to be true when η has been introduced by the application of the freshness rule p_r. In other words, $\eta Ⓡ \phi$ was nested in a freshness quantification on η (which is a variable rather than a constant). In this case the encoding considers the revelation of a restricted node as η using rule p_r. If η does not occur in the interface, then η is not a free name, hence, we introduce it in the interface via rule p_r and proceed as in the first case.

The action modality requires the rule p_π to be applicable. The resulting graph must then satisfy ϕ, but we may need to garbage collected those nodes involved in the synchronization. For that purpose we use four cases (a1-a4).

Finally, consider the encoding of composition. The encoding of the parallel composition P of two processes is done via the parallel composition $\otimes$ of the corresponding graphical encodings. The resulting graph $\llbracket P \rrbracket$ is a tree with the image of p as root, from where several edges depart. Some of them represent subprocesses and the rest correspond to name restrictions. Thus, the encoding of $\phi_1|\phi_2$ is a graph formula that states whether there is a *correct* decomposition of a graph into two components, one satisfying ϕ_1 and the other satisfying ϕ_2 (c4). A correct decomposition requires (c1) to find two complementary and mutually disjoint sets of edges; each set must form (c2) a connected graph including the image of (p) and at least an edge whose source is the image of p; and (c3) any restriction edge has to belong to the right set.

The theorem below states that the proposed encoding is correct.

Theorem 1. *Let P be a process, let Γ be a set of names such that* $\texttt{fn}(P) \subseteq \Gamma$*, and let ϕ be a closed spatial formula. Then, $P \in \llbracket \phi \rrbracket$ iff $\llbracket P \rrbracket_\Gamma \in \llbracket [\phi] \rrbracket$.*

9 Conclusions and Future Work

The paper introduced a graph-based technique for the verification of spatial properties of finite π-calculus specifications. We considered only the deterministic fragment of the calculus, in order to offer as simple a presentation as possible: The choice operator could be included with little effort.

Besides being intuitively appealing, the graphical presentation offers canonical representatives for abstract processes, since two processes are structurally congruent exactly when they are mapped to the same graph wih interfaces (up to isomorphism). The encoding has a unique advantage with respect to most of the approaches to the graphical implementation of calculi with name mobility (such as *bigraphs* [19]): It allows for the reuse of standard graph transformation theory and tools for simulating the reduction semantics of the calculus [15]. Thus, we proposed to check whether a process satisfies a spatial property (formulated in a suitable logic [4]) by encoding spatial formulae in a novel modal graph logic.

With respect to other approaches [2, 12, 21], the main novelty of our logic is a modal operator that binds variables with the items involved in a graph rewriting rule and, in addition, ensures items created by the rule to be new with respect to the environment in which the formula is interpreted. This operator generalizes node quantification, for instance, and this is the key to encode spatial ingredients like revealation of restricted nodes and creation of fresh names.

Besides any consideration on the efficiency and usability of our approach, we believe that a main contribution of our paper is the further illustration of the usefulness of graphical techniques for the design and validation of concurrent systems.

We are planning an implementation of our approach, possibly by extending existing tools for the analysis of graphically designed systems, such as [17, 20].

References

1. P. Baldan, A. Corradini, and B. König. A static analysis technique for graph transformation systems. In K. Larsen and M. Nielsen, editors, *Concurrency Theory*, volume 2154 of *Lect. Notes in Comp. Sci.*, pages 381–395. Springer, 2001.
2. P. Baldan, A. Corradini, B. König, and A. Lluch Lafuente. A temporal graph logic for verification of graph transformation systems. In *Algebraic Development Techniques*, Lect. Notes in Comp. Sci. Springer, 2006. To Appear.
3. R. Bruni, F. Gadducci, and U. Montanari. Normal forms for algebras of connections. *Theor. Comp. Sci.*, 286:247–292, 2002.
4. L. Caires. Behavioral and spatial observations in a logic for the π-calculus. In I. Walukiewicz, editor, *Foundations of Software Science and Computation Structures*, volume 2987 of *Lect. Notes in Comp. Sci.*, pages 72–87. Springer, 2004.
5. L. Caires and L. Cardelli. A spatial logic for concurrency (part I). *Information and Computation*, 186(2):194–235, 2003.
6. L. Caires and L. Cardelli. A spatial logic for concurrency – II. *Theor. Comp. Sci.*, 322(3):517–565, 2004.
7. L. Cardelli, P. Gardner, and G. Ghelli. A spatial logic for querying graphs. In P. Widmayer and F. Trigueiro Ruiz *et alii*, editors, *Automata, Languages and Programming*, volume 2380 of *Lect. Notes in Comp. Sci.*, pages 597–610. Springer, 2002.
8. L. Cardelli, P. Gardner, and G. Ghelli. Manipulating trees with hidden labels. In A. Gordon, editor, *Foundations of Software Science and Computation Structures*, volume 2620 of *Lect. Notes in Comp. Sci.*, pages 216–232. Springer, 2003.
9. A. Corradini and F. Gadducci. An algebraic presentation of term graphs, via gs-monoidal categories. *Applied Categorical Structures*, 7:299–331, 1999.
10. A. Corradini, U. Montanari, and F. Rossi. Graph processes. *Fundamenta Informaticae*, 26:241–265, 1996.
11. A. Corradini, U. Montanari, F. Rossi, H. Ehrig, R. Heckel, and M. Löwe. Algebraic approaches to graph transformation I: Basic concepts and double pushout approach. In G. Rozenberg, editor, *Handbook of Graph Grammars and Computing by Graph Transformation*, volume 1, pages 163–245. World Scientific, 1997.
12. B. Courcelle. The expression of graph properties and graph transformations in monadic second-order logic. In G. Rozenberg, editor, *Handbook of Graph Grammars and Computing by Graph Transformation*, pages 313–400. World Scientific, 1997.
13. F. Drewes, A. Habel, and H.-J. Kreowski. Hyperedge replacement graph grammars. In G. Rozenberg, editor, *Handbook of Graph Grammars and Computing by Graph Transformation*, volume 1, pages 95–162. World Scientific, 1997.
14. M. J. Gabbay and A. M. Pitts. A new approach to abstract syntax with variable binding. *Formal Aspects of Computing*, 13:341–363, 2002.
15. F. Gadducci. Term graph rewriting and the π-calculus. In A. Ohori, editor, *Programming Languages and Semantics*, volume 2895 of *Lect. Notes in Comp. Sci.*, pages 37–54. Springer, 2003.
16. F. Gadducci and A. Lluch Lafuente. Graphical verification of a spatial logic for the π-calculus. In R. Heckel, B. König, and A. Rensink, editors, *Graph Transformation*

for Verification and Concurrency, Electr. Notes in Theor. Comp. Sci. Elsevier, 2006.

17. V. Kozioura and B. König. AUGUR: A tool for the analysis of graph transformation systems. *Bulletin of EATCS*, 87:126–137, 2005. tool available at `http://www.fmi.uni-stuttgart.de/szs/tools/augur`.
18. R. Milner. *Communicating and Mobile Systems: The* π*-calculus*. Cambridge University Press, 1992.
19. R. Milner. Bigraphical reactive systems. In K. Larsen and M. Nielsen, editors, *Concurrency Theory*, volume 2154 of *Lect. Notes in Comp. Sci.*, pages 16–35. Springer, 2001.
20. A. Rensink. The GROOVE simulator: A tool for state space generation. In J. Pfaltz, M. Nagl, and B. Böhlen, editors, *Applications of Graph Transformations with Industrial Relevance*, volume 3062 of *Lect. Notes in Comp. Sci.*, pages 479–485. Springer, 2003. tool available at `http://sourceforge.net/projects/groove`.
21. A. Rensink. Towards model checking graph grammars. In M. Leuschel, S. Gruner, and S. Lo Presti, editors, *Automated Verification of Critical Systems*, volume DSSE–TR–2003–2 of *University of Southampton Technical Reports*, pages 150–160. University of Southampton, 2003.
22. J. Reynolds. Separation logic: A logic for shared mutable data structures. In *Logic in Computer Science*, pages 55–74. IEEE Computer Society, 2002.
23. D. Varró. Automated formal verification of visual modeling languages by model checking. *Software and Systems Modeling*, 3(2):85–113, 2004.